Boolean Logic Simulator in C++

Version 1.0

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 04/08/2024 | 1.0 | Initial Template Filled Out | Kemar Wilson |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

1. Introduction 4

1.1 Purpose 4

1.2 Scope 4

1.3 Definitions, Acronyms, and Abbreviations 4

1.4 References 4

1.5 Overview 5

2. Architectural Representation 4

3. Architectural Goals and Constraints 4

4. Use-Case View 4

4.1 Use-Case Realizations 5

5. Logical View 5

5.1 Overview 5

5.2 Architecturally Significant Design Packages 5

6. Interface Description 5

7. Size and Performance 5

8. Quality 5

# Introduction

HL Group aims to develop a C++ Boolean Logic Stimulator software application that enables users to perform logical operations based on Boolean algebra. This Software Architecture Document provides a comprehensive overview of the architectural design and structure of the Boolean Logic Calculator software. The document outlines the purpose, scope, definitions, acronyms, and abbreviations used throughout the software development process. By detailing the architecture and design principles, this document serves as a reference guide for developers, stakeholders, and users involved in the creation and utilization of the Boolean Logic Calculator software.

## Purpose

# The Software Architecture Document for the Boolean Logic Simulator in C++ serves as a fundamental guide that provides a comprehensive architectural overview of the system. It is designed to capture and convey the significant architectural decisions made during the design and development of the Boolean Logic Simulator. This document utilizes various architectural views to depict different aspects of the system, ensuring a holistic understanding of the architecture.

# Role and Structure of the Document: The primary role of this Software Architecture Document is to outline the architectural design and decisions behind the Boolean Logic Simulator project. It serves as a roadmap for developers, architects, and stakeholders to understand the system's design principles, components, interactions, and constraints. The document is structured to present clear and detailed insights into the system's architecture, emphasizing key design choices and their implications.

# Audience and Usage: The document is intended for a diverse set of audiences involved in the project, including software architects, developers, project managers, quality assurance teams, and stakeholders. Software architects will use this document to define the system's high-level structure, interfaces, and behavior, guiding development efforts. Developers will refer to it for detailed technical information on the system's components and interactions. Project managers can leverage the document to assess project progress and ensure alignment with architectural guidelines. Quality assurance teams will utilize it to define test strategies aligned with the system's architecture. Stakeholders will gain insights into the system's architecture and understand how it aligns with business objectives.

## Scope

The Software Architecture Document for the Boolean Logic Simulator in C++ provides a comprehensive overview of the system's architecture, capturing significant design decisions. It influences system design, development efforts, interactions, scalability, performance, maintenance, and evolution. The document is intended for architects, developers, project managers, quality assurance teams, and stakeholders, guiding them in understanding and utilizing the system's architecture to ensure successful project outcomes.

## Definitions, Acronyms, and Abbreviations

Project/Program: Boolean Logic Simulator in C++ as described by the SDS. See *Section 1.4 References for SDS*

SA/This document: Software Architecture Document. See *Section 1.4 References for SRS*

SDP: Software Development Plan. See *Section 1.4 References for SDP*

SRS: Software Requirements Specifications. See *Section 1.4 References for SRS*

## References

SRS: Accessible at <https://github.com/KNEternity/348HL/tree/main/docs>

SDP: Accessible at <https://github.com/KNEternity/348HL/tree/main/docs>

SA: Accessible at <https://github.com/KNEternity/348HL/tree/main/docs>

## Overview

The **Software Architecture Document** for the Boolean Logic Simulator in C++ serves as a comprehensive guide that delves into the intricate details of the system's architectural design and structure. This document encapsulates a wealth of information, including architectural representation, goals, constraints, use cases, logical views, interface descriptions, size and performance considerations, and quality attributes.

By exploring the **architectural representation**, the document sheds light on what software architecture entails for the current system and how it is visually depicted. It enumerates the necessary views and elucidates the types of model elements contained within each view, providing a holistic understanding of the system's architecture.

The **architectural goals and constraints** section dives into the software requirements and objectives that significantly impact the architecture. It delves into various aspects such as safety, security, privacy, portability, and distribution, capturing the unique constraints that may apply, including design strategies, development tools, team structures, schedules, and legacy code considerations.

In the **use-case view**, significant use cases or scenarios are outlined to highlight central system functionalities or architectural coverage that stress specific architectural points or delicate aspects of the architecture. This section serves as a pivotal point in understanding the system's core functionalities and architectural implications.

Moving on to the **logical view**, the document provides insights into the architecturally significant parts of the design model, emphasizing its decomposition into subsystems and packages. It introduces key classes, their responsibilities, relationships, operations, and attributes, offering a detailed perspective on the system's structural design.

The **interface description** section elaborates on the major entity interfaces, encompassing screen formats, valid inputs, and resulting outputs. It serves as a bridge between the software system and its users, providing clarity on how users interact with the system and what outputs they can expect.

**Size and performance** considerations are meticulously outlined, detailing the major dimensioning characteristics of the software that impact the architecture, along with target performance constraints. This section is crucial in understanding the system's scalability, efficiency, and resource requirements.

Lastly, the **quality attributes** section delves into how the software architecture contributes to the system's capabilities beyond functionality. It emphasizes extensibility, reliability, portability, and other quality attributes, ensuring that the system meets the highest standards in terms of safety, security, and privacy implications.

In essence, the **Software Architecture Document** for the Boolean Logic Simulator in C++ is a comprehensive repository of architectural insights that empowers stakeholders, architects, developers, and project managers to make informed decisions, align with architectural guidelines, and drive successful project outcomes.

# Architectural Representation

The architectural representation section provides a comprehensive insight into the software architecture of the current system, elucidating its essence and visual depiction. It serves as a foundational guide to understanding the architectural design and structure that underpins the Boolean Logic Simulator in C++.

**Software Architecture Overview**: Software architecture for the current system encapsulates the foundational design principles, structural components, and behavioral aspects that define the system's functionality and interactions. It encompasses the high-level blueprint that guides the development and implementation of the Boolean Logic Simulator, outlining the system's key architectural elements and their relationships.

**Representation and Views:** The software architecture is visually represented through a series of architectural views, each offering a unique perspective on the system's design and functionality. These views act as lenses through which different aspects of the system are analyzed and understood.

* **Functional View:** This view focuses on the functional aspects of the system, detailing the primary functions, operations, and interactions that drive the system's behavior. It contains model elements such as use cases, scenarios, and functional requirements, providing a clear insight into the system's functional capabilities.
* **Structural View:** The structural view delves into the system's structural composition, highlighting the components, subsystems, and their relationships. It includes model elements like classes, interfaces, packages, and their dependencies, offering a deep dive into the system's structural organization.
* **Behavioral View:** The behavioral view explores the dynamic aspects of the system, elucidating how components interact and behave at runtime. It contains elements such as sequence diagrams, state machines, and activity diagrams, showcasing the system's behavioral patterns and interactions.

**Model Elements in Each View:**

* **Functional View:** Contains elements such as use cases, scenarios, functional requirements, user stories, and business processes.
* **Structural View:** Encompasses elements like classes, interfaces, packages, modules, relationships, dependencies, and architectural styles.
* **Behavioral View:** Includes elements such as sequence diagrams, state diagrams, activity diagrams, interaction diagrams, and event traces.

By enumerating the necessary views and detailing the types of model elements within each view, the architectural representation section provides a holistic understanding of the system's architecture. It serves as a foundational guide for stakeholders, architects, and developers to comprehend the architectural design and structural nuances of the Boolean Logic Simulator system.

# Architectural Goals and Constraints

[This section describes the software requirements and objectives that have some significant impact on the architecture; for example, safety, security, privacy, use of an off-the-shelf product, portability, distribution, and reuse. It also captures the special constraints that may apply: design and implementation strategy, development tools, team structure, schedule, legacy code, and so on.]

# Use-Case View

[This section lists use cases or scenarios from the use-case model if they represent some significant, central functionality of the final system, or if they have a large architectural coverage—they exercise many architectural elements or if they stress or illustrate a specific, delicate point of the architecture.]

## Use-Case Realizations

[This section illustrates how the software actually works by giving a few selected use-case (or scenario) realizations, and explains how the various design model elements contribute to their functionality. If a Use-Case Realization Document is available, refer to it in this section.]

# Logical View

[This section describes the architecturally significant parts of the design model, such as its decomposition into subsystems and packages. And for each significant package, its decomposition into classes and class utilities. You should introduce architecturally significant classes and describe their responsibilities, as well as a few very important relationships, operations, and attributes.]

## Overview

[This subsection describes the overall decomposition of the design model in terms of its package hierarchy and layers.]

## Architecturally Significant Design Modules or Packages

[For each significant package, include a subsection with its name, its brief description, and a diagram with all significant classes and packages contained within the package.

For each significant class in the package, include its name, brief description, and, optionally, a description of some of its major responsibilities, operations, and attributes.]

# Interface Description

[A description of the major entity interfaces, including screen formats, valid inputs, and resulting outputs. If a User-Interface Prototype Document is available, refer to it in this section]

# Size and Performance

[A description of the major dimensioning characteristics of the software that impact the architecture, as well as the target performance constraints.]

# Quality

[A description of how the software architecture contributes to all capabilities (other than functionality) of the system: extensibility, reliability, portability, and so on. If these characteristics have special significance, such as safety, security or privacy implications, they must be clearly delineated.]